An Optimized Surrogate Based Technique for Reducing the Mutation Test Cost Using Satin Bower Bird Algorihm

Shobana R[a], and Dr Maria priscilla G[b]

[a] Assistant Professor in Department of Computer Applications, Sri Ramakrishna College Of Arts and Science, (Autonomous), Coimbatore, Tamil Nadu, India.
[b]Dean Professor in Department of Computer Science at Sri Ramakrishna College of Arts and Science, Coimbatore, Tamil Nadu, India

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 20 April 2021

Abstract: Fault based testing is also known as mutation testing. In this, the faults are voluntarily introduced in the code to check for the feasibility of the program. If the introduced faults does not produce any variations in the program, it results that the program need debugging for perfect operations. It is a costly test due to its intensive operations in finding the bugs. In order to reduce the expense for mutation testing, automated tools and optimization algorithms were introduced. Surrogate based optimization is also a type of reducing mutation testing cost in all possible ways by using three conditions as objective functions. It able to minimize the test case and achieve high mutation score, but only for few code snipped. To bridge this gap, here an enhanced version of surrogate based optimized is proposed. The existing technique is enhanced by using satin bower bird optimization for efficient test case reduction and kill all the mutants in the snippet. The proposed method is tested on few snippets like calculator, hash table and stack and so on using EmuJava. The proposed optimized surrogate based optimization able to outperform the existing technique by having high mutation score of above 80% for all testing Java snippets. It also able to reduce the computational time and cost by reducing the test cases minimum of 950 for stack snippet and only 50 iterations for processing the CGPA snippet. Therefore, the proposed satin bower bird based surrogate optimization able to reduce the computational cost along with high percentage for killing the mutants. Hence, the proposed approach is best for cost minimization in mutation testing.

Keywords: Fault based testing, Cost minimizationcondition oriented, optimization, surrogate, satin bowerbird, java snippets.

1. Introduction

Fault based testing is also known as mutation testing. In this, the faults are voluntarily introduced in the code to check for the feasibility of the program. If the introduced faults does not produce any variations in the program, it results that the program need debugging for perfect operations.

Mutation testing is a type of costlier fault based testing due to its intensive fault analysis and high computational time. Researchers were proposed different algorithms to reduce the cost and test cases of mutation testing. Some of the algorithms are discussed in following two sections.

Belli et al., (2016) proposed a model based fault detection testing mechanism [1]. Here, a model is designed for generating the test cases. This model helps to analyse the program for possible mutants based on the predefined model. But, it also able to locate the faults in the snipped by block analysis. It is implemented on the graph based dataset. It used the insertion and omission operations for creating the mutant operator and test cases. Constrained test resources and tuning of parameter is required for better operations.

Lima et al., (2016) utilized the higher order mutants as the technique for reducing the cost of mutation testing [2]. Because, in higher order mutant introduces more than one fault at a time and makes difficult for the snippet to kill the mutant. Due to this, it able to utilize lesser test case and produce a bug free. Four types of higher order mutants were utilized to perform the cost reduction process. Secure and bug free snippet avail only if has strong code strength in it.

Jatana et al. (2016) proposed a new approach to reduce the test cases in a test suite [3]. Here, the greedy algorithm is used for reducing the test cases. This approach is implemented on 21 nondeterministic polynomial time based hard problem.

Ganesh (2016) also suggested a solution for nondeterministic polynomial time based hard problem called sequencing problem [4]. Here, the genetic algorithm is applied to find an optimal solution for the problem in lesser time. It is implemented on a process in repair shop. In that, three process cutting, shaping and painting is performed in a sequence manner. It able to reduce the time for inferiority by 100 times as compared to the exhaustive research to reach zero as results. It works effectively for minimal operations.

Gopinath et al. (2016) addressed the short comings of the mutation reduction strategies based on selecting the number of mutants [5]. Here, it analysed the reduction strategies based on number of mutants, effective mutants and its results based on killing the mutants. Most of the techniques were able to reduce the number of mutants. But, it required a proper reduction strategy to produce an effective mutant. Based on that, a theoretical approach is proposed for reducing the mutant is proposed in [5]. Only the practical implementation determine the effect of theoretical reduction strategy on real time dataset.

Jabbarvand and Malek (2017) proposed a mutation testing scheme for the android phone [6]. Here, mostly the mutation testing is performed on a small parts of snippets in an application. Here, the testing is performed with the help of oracle automated tool for testing and it is based on the energy consumption and proper operations of the applications in android phone. Proper measures has to be taken for uninstall an application based on the energy factor alone.

Nishta et al., (2017) performed a survey on different techniques in search based mutation testing operations [7]. It conducted a detailed survey on testing process for fourteen years from 2001 to 2014 and it also include some of the researches from 2015 and 2016 from various journal and website. Based on the data, it discussed about the algorithms most frequently used and effects of each algorithm and most of the algorithms employed genetic algorithm and search based testing for reducing the cost.

Sugave et al., (2017) proposed an optimization approach to perform the white box testing [8]. Here, the optimization process works on both the factors quality and cost minimization of testing. Due to this, it able to reduce the mutants along with the qualified mutant for testing. The diversity dragon fly algorithm were used for this purpose.

Marandi and Khan (2017) proposed a new approach for evaluating the software application [9]. Here, a block by block process is performed to evaluate the software and also determine the fault and rectify it. This approach is able to locate the fault in the block easily to avoid major problems. But, all this process were performed a statically method called defect removal matrices instead of optimization process. It able to produce a perfect bug free application at high computational time.

Kabir et al. (2017) proposed an optimization approach to reduce the duplicates in the testing process [10]. Here, the modified flower pollination algorithm is performed for this process. It able to reduce the duplicates in the test cases but it requires some modifications to minimize the cost for generating the test cases.

Based on the analysis of mutation testing in related works in section 2 and 3, an optimized surrogate optimization using satin bowerbird algorithm is proposed. The explanation of the proposed approach is briefed in section 4. Section 5 highlighted the merits of proposed method based on the evaluation of proposed method results with the existing method. Section 6 and 7 summarize the merits of proposed method and its extension for future works.

2. Related works

In this section, the recent techniques used in the mutation testing cost reduction mechanism is given in table 1.

Table 1. Survey about mutation testing techniques

Author	**Technique**	**Findings**	**shortcomings**
Palomo Lozano et al., (2018)	Integer linear programming	It is mainly focused on performing the mutation testing on the web service oriented applications	Only web service oriented mutation testing is only performed
Ferrari et al., (2018)	Review about cost minimization techniques	Gives ID and paper information about cost minimization techniques	Simple review about cost minimization techniques
Singh and Kumar (2018)	Differential Evolution by Homeostasis Mutation operator	Able to achieve pareto optimization with minimum root mean square error	Tuning of parameters are required.
Ahmed (2018)	Mutation testing and fault prediction	Initially the requirement of testing is discussed Role of mutation testing in developing better software Fault prediction is performed	Complex environment in real time data has to be checked
Madhukar (2018)	Automated mutation testing tool	Swedish ICT company data is used for the observation. MILU tool is used for automatic mutation testing process Analysed with two types of mutation operators: regular and weak	C programming language is highly preferred for better results.

Pizzoleto et al., (2019)	Surveyed about various mutation testing method	First, it classified the mutation testing under seventeen categories. Then, discussed about the type of mutant used in each category	Gives clear information about the mutation testing methods and operator
Rani et al., (2019)	Elitistic genetic algorithm	Selective mutation strategy to introduce low level of faults and mutant generation. It able to identify maximum fault and minimum cost for the test case generation	Redundancy in test cases is the major drawback
Mishra et al., (2019)	Genetic algorithm	First, the path coverage test is performed to remove the duplicate data. Then, the mutation testing is performed to remove the mutants. Due to this, the mutation score is improved.	Chances for removing important information due to removing duplicate information
Bokaei and Keyvanpur (2019)	A complete survey about mutation testing	It classified the mutation testing in three categories automatic testing, cost reduction testing and analytical techniques. Then, the metrics and its working on test case or mutant generation is discussed	It gives an outline about the techniques. But, the issues in the technique is not discussed.
Doliashvili (2019)	Machine learning approach to perform the mutation testing	PIE based analysis for feature extraction on mutation testing. The extracted features are classified using random forest to perform the mutation testing	For java projects, its accuracy is similar to the existing technique named PIE analysis
Gupta et al., (2020)	Detecting software faults in Defects4j repository	Non dominated sorting genetic algorithm-II is applied to optimize the test suite for detecting and locating the software faults	Time and cost reduction is not discussed
Bhatiya (2020)	Optimization approach to reduce the test case generation	Genetic algorithm is used for minimizing the test case generation. It is applied on the Component based system	Cost minimization is not considered
Mohanty et al., (2020)	Optimization technique to reduce the test cases	Ant colony algorithm to minimize the test cases in the suite. It works like a shortest path algorithm for reducing the test cases in the suite	It requires modification for processing larger data to provide reliable results
Taneja et al., (2020)	Machine learning approach for test case minimization	Security is considered as an important factor for reducing the test cases.	This approach can produce an effective result if it consider other parameters also

		Linear regression technique is used for reducing the test case in object oriented software.	
Peng et al., (2020)	Supervised learning approach is used	Auto encoder is used for locating the bugs in the program. It is applied on Defects4j repository	Autoencoder must be combined with SBFL technique to produce best results

3. Conventional Method

The main purpose of software testing is to provide an end user with bug free application. There are numerous types of testing like white box, black box and gray box. Among these, the white box testing is a well-known testing method and expensive method to perform a testing. In this, such a white box testing called Mutation testing is discussed. Since white box testing is an expensive one due to the long term processing of an application and completely provide a bug free application. It requires more financial need and time for performing a mutation testing. In order to compensate the financial and time consumption, the cost minimization concept is introduced in the mutation testing. In previous works, the optimization process like evolutionary algorithms and swarm based algorithms were implemented to reduce the processing time of the testing with an optimal number of test cases. Most of the optimization techniques used the cost for generating the test cases were only considered for the mutant testing with minimal cost utilization. But, it cannot be assured that the cost of test case only provide the sufficient test case to kill a mutant. Hence, to overcome this problem, in existing surrogate based optimization utilized the three parameter to reduce the cost of testing case generation. The three parameters were: 1. whether the case can be attained 2. The control of the variables in the case 3. Necessary of the test case [26]. Based on these factors, the surrogate optimization reduces the number of test cases required for the mutation testing. It performed well in all aspects like reducing overhead, cost and iteration for test case generation. But, it has the following shortcomings as follows:

- The convergence for the optimal test case selection can be reduced better.
- Not able to kill mutants in all program.
- Test cases are high as compared to the random testing algorithms.

These shortcomings of the existing surrogate based approach is overcome with the proposed optimized surrogate based mutant test case minimization is performed in this work. A detail explanation of the proposed method and implementation is given in the further sections.

4. Proposed Method

This section is to describe about the working of the proposed methodology in detail to minimize the mutation test cost. The proposed methodology is the hybrid approach by combining two evolutionary algorithms named satin bower bird optimization and surrogate based optimization. The necessity of this hybrid approach is to achieve the following objectives:

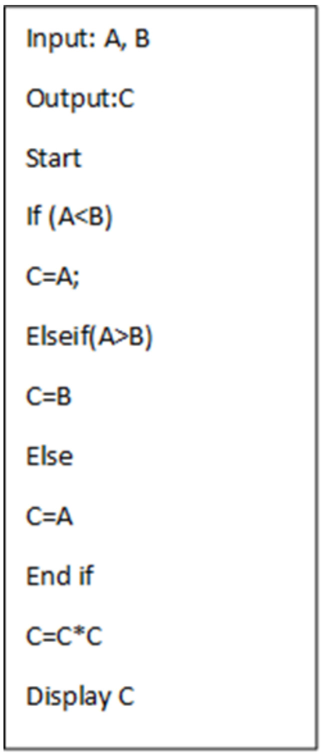
- Able to kill mutants in all programs.
- Reduce the test case and rounds of processing
- Reduce the processing time and cost at minimum level as compared to the existing method.

Generally, the fitness function for mutation testing will be based only on the cost minimization. But, in this it depends on three parameters. Because, the cost alone based minimization not able to kill all the mutants. To overcome this drawbacks, in this the three parameters are used to frame the cost and fitness function for the optimization process.

- Attainable position
- Obligatory position
- Authority position.

The individual role of each position is explained using the square of a bigger number Pseudocode as shown in figure 1.

Figure 1. Square of a bigger number pseudocode



```
Input: A, B
Output:C
Start
If (A<B)
C=A;
Elseif(A>B)
C=B
Else
C=A
End if
C=C*C
Display C
```

4.1 Attainable position:

This is used for evaluating the condition statements in the program. The conditions statements are like if else, switch, while and do- while statements. In this, the condition statement is used in terms of if _else model. Here, the attainable position of variables using if else statement is discussed. This attainable position is also depends on two factors:

- Behavioural position
- Scope of the position

The Attainable positionin terms of nature and coverage of position is defined in formula 1is given using the following equation 1.

$$A_P = \{ B_P, S_P \} \quad (1)$$

4.1.1 Behavioural Position:

Here, the behaviour of the loop condition is analysed. For example, if the loop condition is satisfied it executes its corresponding operations and set the output flag as one otherwise, it set the output flag as zero. The behavioural position is depends on the distance between the conditions in a program. It is denoted in the formula 2.

$$B_P = \{ Distance_{conditions} \} \quad (2)$$

4.1.2 Scope of the Position:

Once, the behaviour of the loop condition is analysed, it check for its validation area based on operations as well as time. Here, the validation is based on the distance between the conditions in the loop. It is given in formula 3.

$$S_P = \{ Ch_P, B\_V_D \} \quad (3)$$

Formula 3 comprises of two elements one is the number of choices〚Ch〛_P in the condition loop and the distance between them as 〚B_V〛_D .

Formula 1 becomes as follows in 4 after substitute the behavioural and scope of the position equations

$$A_P = \{ Distance_{conditions}, Ch_P, B\_V_D \} \quad (4)$$

From the formula 4, it is observed that, in the proposed system, the test cases will be generated for all the possible loop conditions in the snippet to analyse its multiple operations.

4.2 Obligatory position:

Here, the mutants are generated for the functional blocks and the operators in the snippet. Because, the operators and calling a function can change the value of a snippet. Based on this, the formula 5 indicates the obligatory position calculation.

$$O_P = \{ Operators_P, Function_P \} \quad (5)$$

4.2.1 Arithmetic and logical operations:

Here, the mutants are formed by modifying the operators used in the snippet to its opposite sign. Because, the snippet generate different output for different operations. If all the mutant operators produce different output, then the snippet is able to kill all operator mutant. It is defined in formula 6.

$$O_P = \{ALO\ modified\} \quad (6)$$

4.2.2 Functional oriented test cases:

Here, the functional blocks in the snippets are analysed by generating dummy mutant for calling the function in terms of variable names or by passing values. This type of mutants reduce the redundancy of code, if there is an unwanted functional blocks. Formula 7 to indicate the functional

In this, the parameters used in the calling of a function is changed to generate the test cases. It helps to analyse the role of parameters in a calling function. It is given in equation 7.

$$F_P = \{passing\ values\ and\ arguments\ in\ functions\ changed\} \quad (7)$$

The overall obligatory position is given in 8 in terms of ALO and functional test cases,

$$O_P = \{ALO_P, CF_P\} \quad (8)$$

4.3 Authority Position:

This block is to reduce the test case generation for the statements which does not alter its function even after introducing a fault. Due to this, the test cases will not be generated for such a statements. Because such statements cannot be mutated as well as and it cannot be killed. Those statements are indicated No Effect Mutant (NEM) and it is denoted with the help of formula 9.

$$Au_P = \{NEM\} \quad (9)$$

Based on these three positions, the minimization function for surrogate process is in 10

$$S_{OF} = \{A_P, NE_P, Au_P\} \quad (10)$$

4.4 Satin bowerbird based Surrogate optimization implementation using Emujava:

Here, the satin bower bird optimize to minimize the surrogate optimization results. Due to this double optimization process, it able to minimize the number of test cases and also remove the redundant blocks in snippet effectively. The objective of the proposed satin bower bird based surrogate optimization is shown in formula 11

$$SBO_{OF} = Min(S_{OF}) \quad (11)$$

To minimize the above equation 11, the following steps were performed in the satin bowerbird algorithm.

4.4.1 Parameter Initialization:

It is the basic step for all the optimization process to define the problem, iterations, initial solution, updating step size and its boundary.

4.4.2 Bower analysis:

After the first iteration, all the bowers are evaluated to denote its effectiveness in reaching the optimal solution. Here, the analysis is based on attracting the female bower by a male bower. Therefore, the male bowers are only evaluated for this process. Hence, it is denoted as male bower probability (MBP) and it is given in formula 12 and 13.

$$\mathrm{MBP_i} = \frac{SBO_\mathrm{i}}{\sum_{\mathrm{n}=1}^{50} \mathrm{f_n}} \quad 12$$

$$\mathrm{SBO_i} = \begin{cases} \dfrac{1}{1+\mathrm{SBO}(\mathrm{x_i})}, & SBO(\mathrm{x_i}) \geq 0 \\ 1 + |\mathrm{SBO}(\mathrm{x_i})|, & \mathrm{SBO}(\mathrm{x_i}) < 0 \end{cases} \quad 13$$

4.4.3 Bower evaluation:

Once, the probability of bower is analysed and all the bowers are subjected to finding the optimal solution for the problem in 11.

4.4.4 New solutions:

The position of bower needs to update regularly for finding better solution, but the updating its position is also within its limit. Hence, to limit and update position, the roulette wheel and MBP attractions are used as dominant factor. The following formula 14 and 15 is used for the position of bowers.

$$NP_{ik}^{n} = NP_{ik}^{o} + \mathrm{MBPA}_{k}\left(\left(\frac{NP_{jk} + NP_{elite,k}}{2}\right) - NP_{ik}^{o}\right) \quad 14$$

$$MBPA_{\mathrm{k}} = \frac{0.94}{1 + MBP_{\mathrm{j}}} \quad 15$$

1.1.1 Reducing bowers:

The bower with lower MBPA (i.e attractiveness) are removed to increase the chance for the other bower and also to faster the operations. It is given in formula 16 to 18.

$$\sigma = \mathrm{z} \times (\text{minimal test cases} - \text{maximal test cases}) \quad 16$$

$$\mathrm{NP}_{\mathrm{ik}}^{\mathrm{n}} \sim \mathrm{S}(\mathrm{NP}_{\mathrm{ik}}^{\mathrm{o}}, \sigma^2) \quad 17$$

$$\mathrm{S}(\mathrm{NP}_{\mathrm{ik}}^{\mathrm{old}}, \sigma^2) = N\mathrm{P}_{\mathrm{ik}}^{\mathrm{o}} + (\sigma \times \mathrm{S}(0,1)) \quad 18$$

The percentage between the difference of minimal and maximal test cases is given as z.

1.1.2 Minimized Test cases:

The final minimal test cases for the mutation testing is determined by double optimization process satin bower bird and surrogate optimization after it complete all the iterations.

1.2 **Algorithm**:

The workflow diagram and steps involved in the proposed methodology is explained below.

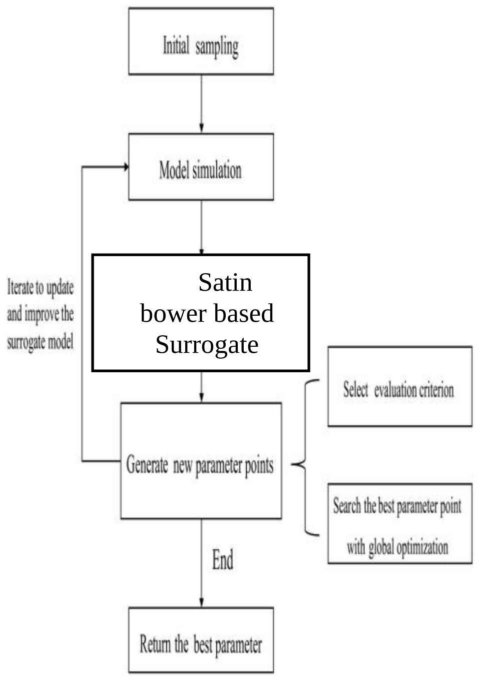

Initial sampling
Model simulation
Iterate to update and improve the surrogate model
Satin bower based Surrogate
Select evaluation criterion
Generate new parameter points
Search the best parameter point with global optimization
End
Return the best parameter

Figure 1. proposed SBO-surrogate test case minimization

The working of proposed method is shown in pictorial format in figure 1 and the steps are given below.

```
Input: snippets, SBO initialization parameter
Output: Minimized test cases
Start
Initialize bower parameter and solutions
Calculate test cases based on equations 1 to 11
Determine best bower
While (iterations< max iterations)
Bower probability based on equation 12 and 13
For all bower
    For all bower elements
Update bower positions using equations 14 to 18
End for
End for
```

```
Calculate best bowers
Calculate the elite bowers
End while
Return optimal test cases
```

5. Results & Discussion

This section is to discuss about the performance of the proposed method in minimizing the cost of mutation testing. For the implementation, the Java software and application called emuJava is used. The application is modified with our proposed approach and it is denoted as emuJava version 4.0.

The version 4.0 indicates the mutation testing performed with the help of satin bower bird based surrogate optimization. To verify the proposed method performance, the mutation testing is performed on the following snippets in table 2.

Table 2. Java snippets

• Auto door • Hash table • Stack • CGPA calculator	• Calculator • Triangle • Binary search tree.

The satin bowerbird based surrogate optimization created all the possible test cases for the snippets in table 2. An evaluation metric is needed to analyse the performance of a method. Hence, in this work, the following two metrics were used for the evaluation of the proposed mutation testing method. The evaluation metrics are as follows:

- Test case generation
- Average Number of kill of mutant

5.1 Test case generation:

Test cases are important for the mutation testing to evaluate the performance of proposed testing method. An algorithms should use lesser number of test cases and rounds to kill all the mutants in the program. The comparison of the test case for different approaches is shown in table 3.

Table 3. Test case comparison

Approaches	**Random testing**		**Improved Genetic algorithm**		**Surrogate based optimization**		**Satin bowerbird based surrogate optimization**	
Program	**Rounds**	**Test cases**	**Rounds**	**Test cases**	**Rounds**	**Test cases**	**Rounds**	**Test cases**
Autodoor	310	7750	140	3500	135	7751	125	7000
Hash table	1052	15375	289	7225	280	8500	255	8000
stack	91	26300	208	5200	200	1000	190	950
CGPA calculator	380	2325	74	1850	65	2000	50	1957
calculator	130.93	3250	157	3925	150	4000	135	3896
triangle	-	10800	373	9325	370	9400	345	9200
Binary search tree	263	6575	255	6375	350	6450	325	6250

Table 3 shows the number of test cases required by each algorithm to kill the mutants. Among these algorithms, the proposed satin bower bird based surrogate optimization is best in terms of using lesser test cases to kill the mutants as compared to the existing techniques like random testing, improved genetic algorithm and surrogate based optimization.

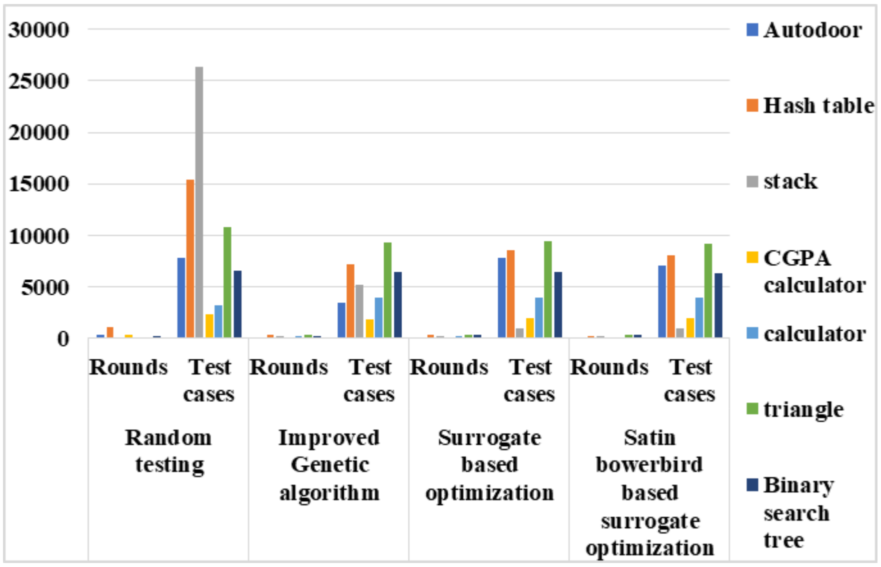


Figure 3. Test case comparison

Figure 3 shows the comparison of the test cases used and rounds involved in each algorithm to kill the mutant. In this also, the satin bower bird based surrogate optimization utilize minimum number of rounds to kill the mutants as compared to the conventional approaches

5.2 Average Number of kill of mutant

The term average number of kill of mutant is to indicate the successful number of mutants killed in the proposed method. Because, the higher number of kill mutants only provide a bug free process. It is otherwise called as mutation score. The formula is used to calculate the average number of killed mutants (ANKM)

$$ANKM = \frac{killed\ mutants}{total\ mutants} \tag{11}$$

The comparison of the mutant score with the proposed and existing techniques are shown in the table format in table 4 and pictorial format in figure

Table 4. ANKM comparison

Program	Random Testing (%)	Improved genetic algorithm (%)	Surrogate based optimization (%)	Satin bowerbird based surrogate optimization
Autodoor	77	82	90	92
Hash table	68	74	80	85
stack	69	85	89	93
Cgpa calculator	96	76	82	86
calculator	95	86	91	94
triangle	79	50	52	82
Binary search tree	87	74	78	91

The comparison of mutation score is shown in the table 4. It is observed that the proposed method has higher number of mutants killed as compared to the existing methods by having high mutation score. The visualization of the mutant score comparison is shown in figure

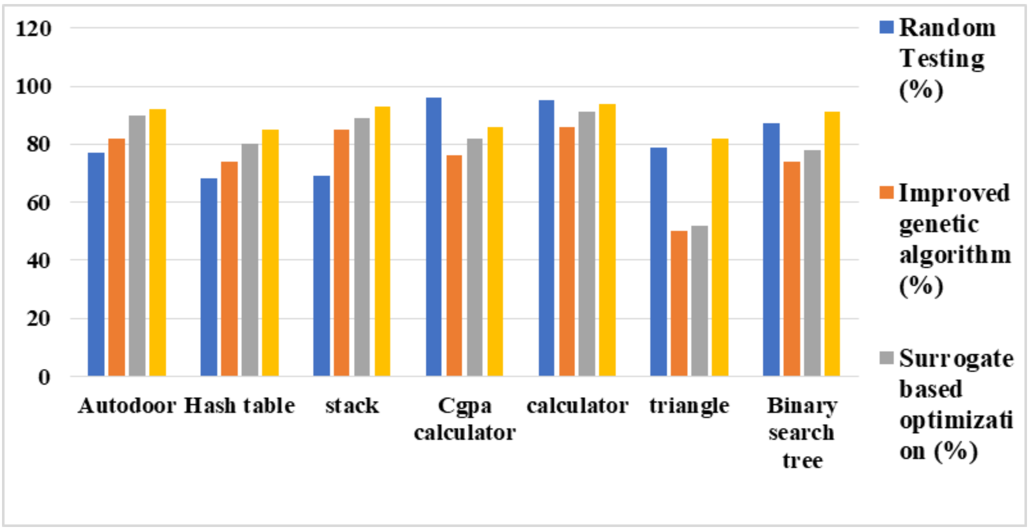


Figure 4. ANKM comparison

It is observed from the above visualization also, the proposed satin bowerbird based surrogate optimization is having high mutant score in all programs as compared to the existing methods like improved genetic algorithm and surrogate based optimization. Because, in both the existing methods the higher mutation score is not obtained for all the cases as like in the proposed approach. Therefore, the proposed approach outperforms the existing techniques and best for the minimization of mutation testing cost.

Based on the above evaluations, it is observed that the proposed satin bowerbird based surrogate optimization is having high mutant score in all programs with lesser number of test cases and rounds to kill the mutants. While in existing approaches, the mutation score is not improved even with the more number of processing and test cases. Hence, the proposed satin bowerbird based surrogate optimization is best for the mutation test cost minimization.

6. Conclusion

The main purpose of software testing is to provide an end user with bug free application. There are numerous types of testing like white box, black box and gray box. Among these, the white box testing is a well-known testing method and expensive method to perform a testing. In this, such a white box testing called Mutation testing is discussed. Mutation testing requires more financial need and time to provide a bug free program. In order to compensate the financial and time consumption, the cost minimization concept is introduced in the mutation testing using optimization techniques. Most of the optimization techniques used the cost for generating the test cases were only considered for the mutant testing with minimal cost utilization. This concept is replaced by using three important aspects in test case generation to perform mutation testing with minimal cost in the proposed satin bowerbird based surrogate optimization. It outperform the existing techniques like random testing, improved genetic algorithm and surrogate based optimization due to the following reasons:

- Lesser number of test cases
- Higher mutant score
- Lesser number of iterations to kill all mutant
- Faster computation time.

7. Future work

In future, the proposed method is modified by implementing different optimization algorithm to improve the mutation score with lesser number of test cases.

References

1. Belli, F., Budnik, C. J., Hollmann, A., Tuglular, T., & Wong, W. E. (2016). Model-based mutation testing—approach and case studies. Science of Computer Programming, 120, 25-48.
2. Lima, J. A. P., Guizzo, G., Vergilio, S. R., Silva, A. P., Filho, H. L. J., &Ehrenfried, H. V. (2016, September). Evaluating different strategies for reduction of mutation testing costs. In Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (pp. 1-10).
3. Jatana, N., Suri, B., Kumar, P., &Wadhwa, B. (2016, March). Test suite reduction by mutation testing mapped to set cover problem. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (pp. 1-6).
4. Ganesh, J. (2016). Performance Evaluation of Genetic Algorithm on a Setup Cost Minimization Problem.
5. Gopinath, R., Alipour, M. A., Ahmed, I., Jensen, C., &Groce, A. (2016, May). On the limits of mutation reduction strategies. In Proceedings of the 38th international conference on software engineering (pp. 511-522).

6. Jabbarvand, R., &Malek, S. (2017, August). μDroid: an energy-aware mutation testing framework for Android. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 208-219).
7. Nishtha, J., Bharti, S., & Shweta, R. (2017). Systematic literature review on search based mutation testing. e-Informatica Software Engineering Journal, 11(1).
8. Sugave, S. R., Patil, S. H., & Reddy, B. E. (2017, June). DDF: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing. In 2017 International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 701-707). IEEE.
9. Marandi, A. K., & Khan, D. A. (2017). An Approach of Statistical Methods for Improve Software Quality and Cost Minimization. International Journal of Applied Engineering Research, 12(6), 1054-1061.
10. Kabir, M. N., Ali, J., Alsewari, A. A., &Zamli, K. Z. An adaptive flower pollination algorithm for minimizing software testing redundancy.
11. Palomo-Lozano, F., Estero-Botaro, A., Medina-Bulo, I., &Núñez, M. (2018, July). Test suite minimization for mutation testing of WS-BPEL compositions. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 1427-1434).
12. Ferrari, F. C., Pizzoleto, A. V., & Offutt, J. (2018, April). A systematic review of cost reduction techniques for mutation testing: preliminary results. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 1-10). IEEE.
13. Singh, S. P., & Kumar, A. (2018). Multiobjective differential evolution using homeostasis based mutation for application in software cost estimation. Applied Intelligence, 48(3), 628-650.
14. Ahmed, I. (2018). Improving the Quality of Software Using Mutation Testing and Fault Prediction.
15. MADHUKAR, E. (2018). Generate Test Selection Statistics with Automated Mutation Testing.
16. Pizzoleto, A. V., Ferrari, F. C., Offutt, J., Fernandes, L., & Ribeiro, M. (2019). A systematic literature review of techniques and metrics to reduce the cost of mutation testing. Journal of Systems and Software, 157, 110388.
17. Rani, S., Suri, B., & Goyal, R. (2019). On the effectiveness of using elitist genetic algorithm in mutation testing. Symmetry, 11(9), 1145.
18. Mishra, D. B., Mishra, R., Acharya, A. A., & Das, K. N. (2019). Test data generation for mutation testing using genetic algorithm. In Soft Computing for Problem Solving (pp. 857-867). Springer, Singapore.
19. Bokaei, N. N., &Keyvanpour, M. R. A Comparative Study of Whole Issues and Challenges in Mutation Testing. In 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI) (pp. 745-754). IEEE.
20. Doliashvili, N. Predicting Survived and Killed Mutants.
21. Gupta, N., Sharma, A., &Pachariya, M. K. (2020). Multi-objective test suite optimization for detection and localization of software faults. Journal of King Saud University-Computer and Information Sciences.
22. Bhatia, P. K. (2020). Test Case Minimization in COTS Methodology Using Genetic Algorithm: A Modified Approach. In Proceedings of ICETIT 2019 (pp. 219-228). Springer, Cham.
23. Mohanty, S., Mohapatra, S. K., &Meko, S. F. (2020). Ant Colony Optimization (ACO-Min) Algorithm for Test Suite Minimization. In Progress in Computing, Analytics and Networking (pp. 55-63). Springer, Singapore.
24. Taneja, D., Singh, R., Singh, A., & Malik, H. (2020). A Novel technique for test case minimization in object oriented testing. Procedia Computer Science, 167, 2221-2228.
25. Peng, Z., Xiao, X., Hu, G., Sangaiah, A. K., Atiquzzaman, M., & Xia, S. (2020). ABFL: An autoencoder based practical approach for software fault localization. Information Sciences, 510, 108-121.
26. Bashir, M. B., & Nadeem, A. (2017). Improved genetic algorithm to reduce mutation testing cost. IEEE Access, 5, 3657-3674.
27. Moosavi, S. H. S., &Bardsiri, V. K. (2017). Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation. Engineering Applications of Artificial Intelligence, 60, 1-15.
28. Han, Z. H., & Zhang, K. S. (2012). Surrogate-based optimization. Real-world applications of genetic algorithms, 343.